Insider's Guide to the iSeries Developer Roadmap

One of the biggest challenges to creating modern business applications, whether modernizing legacy systems or creating new applications from scratch, is being able to tell the difference between the technology that is going to take you the distance, and those methods and products that will leave you stalled on the techno-evolutionary chain.

To help developers avoid those pitfalls, IBM came up with a comprehensive plan called the iSeries Developer Roadmap. It is designed to assist developers as they migrate from traditional text-based applications, taking them through a series of five consecutive development stages… to bring them, ultimately, to fully portable and scalable Java-based Web applications.

This white paper will give an insider's view into this modernization plan. In each section, we'll start by gaining a better understanding of each of the individual stages of IBM's roadmap, the benefits of each stage, IBM's recommended methods to get there, and then a realistic view of what that will mean for developers.

Applying IBM's iSeries Developer Roadmap:

There are five chronological levels or stages to the iSeries Developer Roadmap (**Figure 1**):

1. Improve Your Productivity
2. Enhance the End User Experience
3. Create a Modular Architecture
4. Integrate Applications
5. Integrate Business Processes.

IBM Stage 1: Improve Your Productivity

Formerly known as the "Better Tools" roadmap stage: "The first step in the iSeries Developer Roadmap does not involve any change to the application code in use today. Rather, it enables the replacement of traditional development tools with more exciting and modern tools that support the same code base.

Ultimately, applications will have the option of being written in modern or traditional languages, such as Java, RPG or COBOL.

This flexibility will allow continued utilization of green-screen user interfaces through DDS or development of new and next generation interfaces." [i]

Benefits of Improving your Productivity:

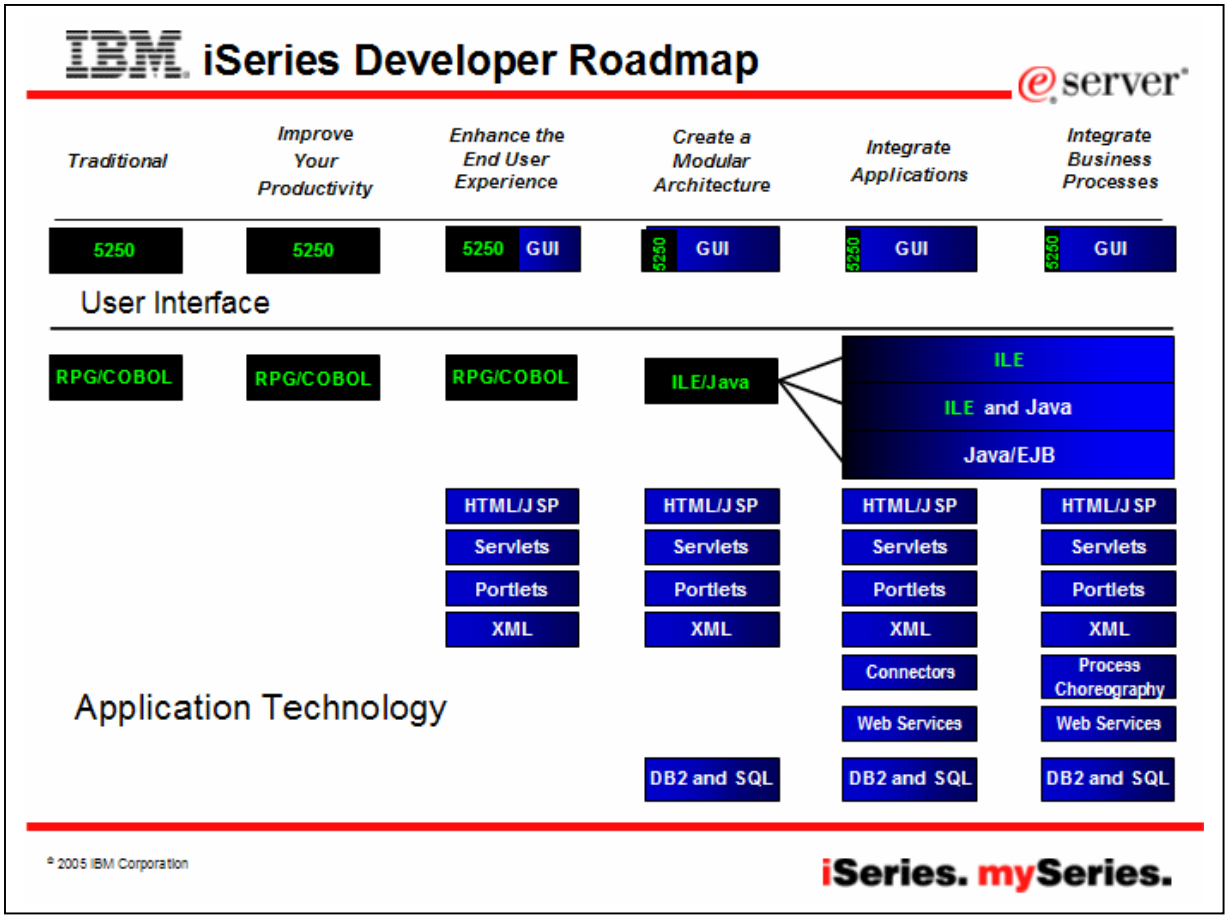
Learned skills: This first stage helps developers learn to create applications more efficiently and to become more accustomed to using modern tools and technologies.

Changing minds: Ultimately, it serves as a method to begin changing developer hearts and minds to get them thinking differently about how applications are created. Although this stage is not centered around applications as the latter stages are, it sets the crucial tone of forward motion… and a commitment to modernization.

IBM's recommendation:

IBM's recommendation of tools in this section deal primarily with those tools able to accommodate text-based applications that are written in RPG and COBOL. Specifically, in much of this stage of the roadmap, IBM recommends its own WebSphere Development Studio client (WDSc). As it is IBM's roadmap, and IBM's product, this is not terribly surprising. However, WDSc has its own pros and cons, which should be considered.

Figure 1: *This is IBM's diagram of the roadmap, showing first, the type of interface, the underlying technology and some of their own recommended tools used to achieve each stage.*



Benefits to using WDSc:

Price is right: It is free to those on current IBM maintenance programs.

Consistent environment: This Integrated Development Environment, or IDE, gives users a familiar development interface to work with when moving to unfamiliar platforms later on.

Longevity: IBM's sponsorship speaks to its survival through subsequent technology stages.

Drawbacks to WDSc:

Tool learning curve: An integrated development environment (IDE), WDSc is reputed to have a fairly steep learning curve, and requires more training than is generally publicized.

In fact, iSeriesNetwork.com has even devoted a special blog [ii] documenting the personal experiences of five developers and their struggles to get the hang of it. It is not a simple plug and play solution.

Language learning curve: Although this IDE environment, once learned, will remain familiar through later stages, it still requires developers to manually code all their programs. That means factoring in the additional learning curve of all subsequent development languages to be used, like Java.

Estimates vary with regard to Java's learning curve, but sources generally gauge it between six months to two years leaning toward the higher end of that for a solid understanding and application of its principles.

Hardware and maintenance issues: Perhaps the largest drawback is that it is a client-based development tool, and it is recommended that PCs running it have at least 512MB of memory.

'Improved Productivity stage' sets the crucial tone for forward motion.

However, Sharon Hoffman's white paper on the subject contradicts this claim and states that anything less than 1GB will cause problems… therefore PC upgrades are recommended. [iii]

And, there may be iSeries upgrades necessary as well.

Additionally, its client-side residence means maintenance becomes more troublesome. That can get pretty pricey for an otherwise "free" solution. Keep in mind, WDSc is just one option.

There are many different kinds of tools available in application development… ranging from IDEs like WDSc to 4GLs to modern menu-based smartTools like m-Power™ that you can actually teach to develop for you. These tools vary… in price, in performance, in capabilities, and in learning curve, not to mention platform.

IBM provides a listing of recommended Roadmap tools on its Web site at: http://www-1.ibm.com/servers/eserver/iseries/roadmap/tools.html

IBM Stage 2: Enhance the End User Experience

Formerly known as "Better User Interface," according to IBM this second step of the roadmap: "Gives them the tools and process to take their application presentation, to the next level with a browser, client-server, or even a mobile device using pervasive technologies." [iv]

So, this step is about finding a way to create a Graphical User Interface without touching the underlying green screen application… in other words, screen scraping, or IBM's Web-facing.

Benefits to Enhancing the End User Experience:

There are many general benefits realized from just "Web-ifying" text-

based applications to quickly create a better user interface. These include:

Improved accessibility: It gives users, who may be remote, secure access to mission-critical information. For example, a traveling sales representative can greatly improve order processing, customer service, and logistics for his client right on the spot.

Also, the ability to give partners or customers secure access via the Web to easy-to-use account maintenance applications and electronic invoicing provides the additional value of removing extraneous work burdens from employees.

Inherent added value: Creating graphical elements, and linking to non-text-based features, like product photos, and even how-to-videos demonstrating the use of products, etc, can add immeasurable value for Web site visitors.

Additionally, these features also prevent errors, such as placing an order for the wrong item.

Saved training dollars: Green-screen or text-based applications have long been a thorn in the side of department managers when it comes to training a new employee, as they are not always easy to use or particularly intuitive.

However, users find Web-based applications much easier to grasp as the majority of employees have at least had some exposure to the Internet. That means shorter training cycles, and less rookie mistakes.

IBM's recommendation:

IBM's recommendation of tools in this stage primarily fall into the category of IBM's "Web-facing" tool or use of a screen-scraper program.

> Acknowledging that applications are outdated is the first step to change.

Positives to Web-facing or screen scraping:

Experience: Developers unfamiliar with the Web world will gain new experience with HTTP, WebSphere, HTML, and firewalls.

Low risk: There's a seemingly low risk of losing a lot of money because Web-facing is billed as free with no need for new developers. And, there is no analysis work, so that means a faster timeframe to bring these applications to the Web.

Emotional first steps: It's an emotional comfort for developers to have a Web look and still have total control over their code. And, acknowledging that applications are outdated serves as the first step to change.

IBM's recommendation could serve as a good stopgap measure. For example, let's say you were opening up several remote distribution warehouses across the country, and you just wanted to give these warehouses access to an inventory lookup application. You could just use a screen scraper or Web-facing to create a graphical application those users could access from a Web browser wherever they are.

Now, while this is fine as a quick-fix, it's important to realize that often six months to a year later, the users will be unhappy with the solution, and it will end up back on IT's list for re-engineering. Why?

Here are some of the negatives:

Lacking interface: Because the Graphical User Interface or GUI is just a graphical representation of the text-based application below its surface, it is generally not very intuitive or "Web-like." It will often include elements that are either extraneous or nonsensical in the context of a real Web-based application. (See **Figure 2**)

So, what seems like a "quick-fix" can often require extensive painting or redesign to remedy the situation.

Slow application performance: Because applications have actually added another layer on top of an already running RPG or COBOL program, they can run relatively slowly.

Slow system performance: IBM's Web-facing solution runs on the WebSphere Application Server, which must remain on the iSeries. This, in congress with the extra application layer creates extra load on the iSeries, which translates to slowdowns for all your applications.

IBM Stage 3: Create a Modular Architecture

Formerly known as the "Better Architecture" stage, according to IBM, this involves: "Separation of user presentation, business logic, and database access creating reusable application components, callable interfaces, and database callable modules." [v]

Examples of this include referential integrity and stored procedure types of function… By re-architecting the application into a modular one, you also allow for the replacement/ addition of modern technologies such as browser-based interfaces and distributed database activity.

That "modularization" of the code they refer to means specifically splitting apart user interface from the business logic and isolating functions such as database access.

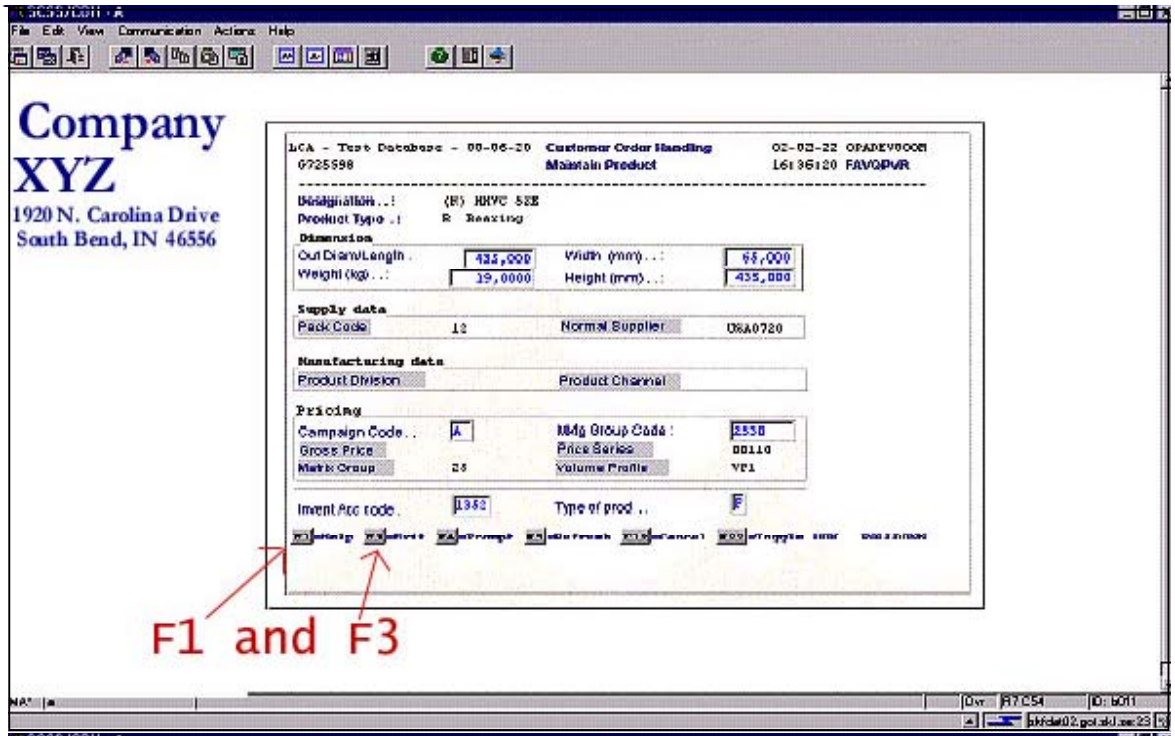
This can also be called **n-tier architecture**.

Traditionally, applications were built as one long block of code with multiple screens within each program, and complicated calls within the code.

One reason for designing applications this way was that processors originally were able to process the code within one longer program at a much faster speed than opening and closing many individual smaller interconnected programs.

Today, that is no longer the case.

Figure 2: *This is an example of a screen-scraped application. It looks like a text-based application that has been placed on a Web site. There are no dropdown menus, or radio buttons, or interactive features outside of entry fields, and it even includes text commands like F1, F3, and F6—that are made into graphic buttons, but these are nonsensical on the Web.*



Instead, the recommended architecture is to modularize the code within each program, and integrate multiple individual programs into larger application systems. Think of it as one screen per program.

Benefits to Creating a Modular Architecture:

Improved maintenance: Sharon Hoffman, in her white paper, *An Independent Analysis of the iSeries Developer's Roadmap* elaborates on one benefit: "Better architecture proposes a structure that uncouples business logic from user interfaces, thus making it possible to modify user interfaces without affecting business logic and vice versa." [vi]

This also means pieces can be independently maintained by in-house experts to each component. For example, a Web designer may want to update or develop the look for the GUI or presentation layer, while a systems analyst might be the go-to person for the backend logic.

Save money by cutting development time: Reusing pre-built modular components means faster development and a shorter testing phase which translates to a faster speed-to-market.

Easier troubleshooting: Pinpointing development errors or bugs becomes easier when re-using pre-built and pre-tested components, because it narrows the scope.

IBM's recommendation:

IBM recommends sticking with manual development in RPG or COBOL, and using these languages to begin breaking applications up and restructuring using programming techniques such as stored procedures.

> By re-architecting the application into modules, you allow for the addition of modern technologies such as browser-based interfaces and distributed database activity.

The positive to IBM's recommend-ation is that you are working with familiar languages to create the underlying business logic, reducing the immediate learning curve for creating new application designs.

However, the problem with IBM's recommended method is that it is difficult to manually restructure major applications into new architectures. It's also going to take you a long time to complete the process, which doesn't make sense when the same results can be achieved using a more comprehensive development tool, allowing users to advance through this stage at a much faster rate.

IBM Stage 4: Integrate Applications

This stage promotes Java adoption.

Formerly "Better Portability," according to IBM, "This step along the Roadmap is designed for those who prefer that their application, or pieces of their application have the ability to deploy and execute on multiple platforms. This step is also for those shops that might have a requirement to integrate some Java components into their traditional applications." [vii]

This stage, in a nutshell, involves a move from creating business logic in traditional languages like RPG or COBOL to begin writing it in Java.

Portable applications are server and vendor neutral, making it much easier to use applications to solve a wider range of business problems than ever before with optimum utilization of development resources. Portability is important in today's development world in several ways.

Benefits to Integrating Applications:

Control system and hardware costs: You can run different modules

on different hardware for dynamic load balancing, which means performance bottlenecks can quickly be remedied.

For example, the server process could be moved to a faster server at runtime. No longer would you be stuck on outdated hardware, or underperforming systems. Instead, you can distribute information the way that is most cost-effective for you.

Integrate disparate databases: Access data from any and all disparate database systems simultaneously. Perfect for companies involved in recent mergers or acquisitions with varying underlying business systems.

Forward flexibility: Applications, once created, can be moved to any Web serving platform without redevelopment.

IBM's recommendation:

IBM recommends you learn Java, and to use Java, Web, and Web Services tools in the WebSphere Developer Studio client or WDSc. They also recommend you use simple, standard Java—referred to as Java 2 Standard Edition (J2SE)—that accesses data in the familiar SQL ways.

You must not only know Java, but you must also have a general understanding of object oriented analysis and design in order to write Java business logic.

Hoffman's *Independent Analysis* says, "For many iSeries shops, there's no particular reason to implement Better Portability." And that this is only necessary, "…if you have a requirement to deploy applications to multiple platforms, then Better Portability represents the next step in the evolution of iSeries development." [viii]

Better Portability should be on the radar of every developer out there, *especially* those in small to medium shops.

That is not necessarily true. Better portability should be on the radar of every iSeries developer out there, especially those in small to medium shops. Why? Because portability gives you the freedom to control your application deployment, your hardware, your memory resources and load balance, your vendors, your maintenance obligations, and therefore your own costs.

That said, this phase is probably most immediately important for software vendors that want to be able to sell their business applications or other software products to run on any platform, and access any database… thereby expanding their customer base and market reach.

IBM Stage 5: Integrate Business Processes

IBM originally described its final stage as Better Scalability. "Systems and applications working together, connecting with suppliers and customers to achieve process efficiencies, better communications and improved service."

The focus, then, here is on creating applications that can adjust quickly to changes and fluctuations on both the user side, and on the information side.

Generally, this is for those companies who are rapidly growing, or businesses hit with seasonal spikes that place too much demand on their systems.

Benefits to Integrated Business Processes

Better response time: Applications only take on the resources that they need at any given point, regardless of data transaction, improving speed.

Better reliability: Administrators can rest easy knowing that a surprise spike in users, or a bump in files coming through the system, won't bring business to a grinding halt.

Better customer and departmental relations: Users logging on during a high demand time will receive the same performance as those logging on during a lower usage time, building trust and confidence in both company and system.

Happy users are good for business… and like it or not, that's not conjecture anymore.

An ongoing study by the Forum for People Performance Management and Measurement, based at Northwestern University, recently broke new ground by focusing on employees who do not have direct contact with customers, and quantifying the results.

They found a one-unit increase in employee satisfaction led to a 0.31-unit increase in customer satisfaction. In turn, a one-unit increase in customer satisfaction created a 0.28-unit improvement in financial performance. [ix]

IBM's recommendation

IBM says that in order to master this stage, you must use the Advanced WebSphere Application Server administration, and understand Enterprise JavaBeans (EJB) development, performance tuning, and clustering.
A big concern about this stage is that IBM expects developers to rewrite J2SE (remember, Java 2 Standard Edition?) applications and turn them into J2EE (Java 2 Enterprise Edition) applications.

That is because J2EE can use Enterprise JavaBeans, and has more extensive capabilities. But, that also means re-working the applications that it has now taken months or years to create manually. That's unrealistic unless developers choose their tools wisely.

Using the Roadmap

In the end, every business owns an individual set of challenges, problems, and solutions on the road to modernization. Some businesses are way ahead of the game with Web systems in place and elaborate intranets and extranets, while other businesses may have half of their applications fully modernized leaving the other half to remain in RPG for years to come, and still others are starting from scratch.

And, because each company and developer is unique, the timeline below caused some spirited debate around the halls of mrc. It is just so difficult to project a universal timeline when it comes to development.

And obviously even a stage as simple as Better Tools could take 6 weeks, 6 months, 2 years… depending on the person researching the tools, compiling needs, development backlogs, and the like.

In using a general scale of 1000 programs or screens, our experts compared one traditional iSeries programmer following IBM's roadmap to the letter, and using IBM's methods to manually develop all of their solutions, versus that same programmer using m-Power™ instead. (**Figures 3 and 4**)

In Figure 3, we see IBM's iSeries Developer Roadmap depicting how m-Power™ fits in, and where it saves the most time. The timeline is set in developer-months, meaning one traditional iSeries developer devoting 100% of his or her time.

In Figure 4, we see the iSeries Developer Roadmap with each stage and the months it would take to accomplish each manually, if he or she were to follow the Roadmap to the letter.

As you can see from the graphics, there are major time savings in applying m-Power™ to the iSeries developer roadmap, in general. But there's a way for your work to begin paying off even faster using the iSeries Developer roadmap with m-Power™.

Improving the ETA of your ROI.

The biggest advantage to using a tool like m-Power™ within the roadmap, rather than following the roadmap manually, is the early point at which

you can begin enjoying the most benefits

Even if you're not about to address 1000 programs all at once… you probably do have a pressing need to update a handful at a time. For example, let's say you wanted to update 25 programs for your customer service department. Manually adopting each roadmap stage would mean months of work.

However, using m-Power™, means that as soon as you check off your application specifications in its Web menu, and click a build/compile button, you will immediately produce a Java application that is fully portable to any platform or database you choose.

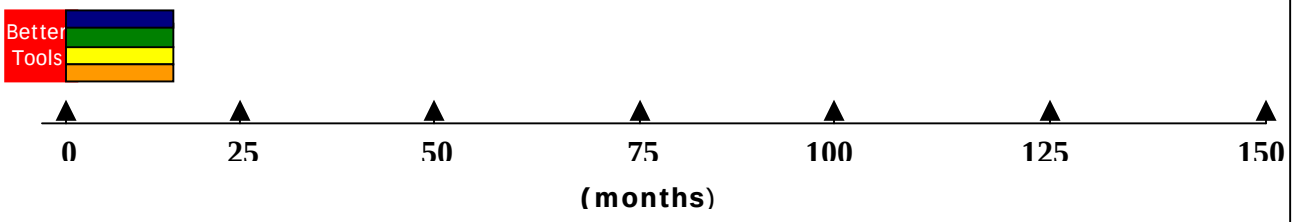
Just look at the timelines. In the same time that you are still working on screen-scraping applications in the better user interface stage, you could already have created a fully portable Java servlet application, that runs on any platform and accesses any database.

If you would like to see a case study demonstrating these methods applied in real life, read on. The second part to this white paper, m-Powered™: ***A practical application of the iSeries Developer Roadmap***, gives a blow-by-blow account of the real-life story of how the mrc-Productivity Series, an RPG-based system of iSeries programs became m-Power™, a graphical Java-based independent solution… and how one developer did it in record time.

If you'd like more information on the iSeries Developer Roadmap, you can visit IBM's Web site at: **http://www-1.ibm.com/servers/eserver/education/cust/iseries/paths/j2ee/ests_isli_j2ee_index.html**

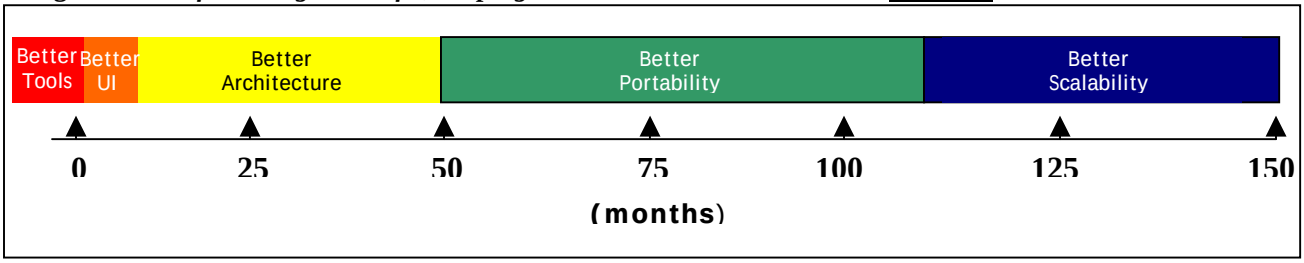
Or, Sharon Hoffman's white paper: "An Independent Analysis of the iSeries Developer's Roadmap" is also very helpful. It can be found at: **ftp://ftp.software.ibm.com/common/ssi/rep_wh/n/ISW00197USEN/ISW00197USEN.PDF**

Figure 3: ***Timeframe for 800 of 1,000 total programs or 80%* modernized using m-Power™***



**m-Power™ takes care of at least 80% of a business's total modernization needs. That means 800 of 1,000 programs would be fully portable and written in Java J2EE. . (Most businesses, in fact, would actually be able to modernize more like 90-95% of their total applications this quickly.) The rainbow stripes in the m-Power™ diagram represent the roadmap's Better user interface, Better architecture, Better portability and the J2EE part of Better scalability m-Power™ provides.*

Figure 4: ***Timeframe to get 800 of 1,000 programs or 80%* when modernized manually.***



m-Powered™: A practical application of the iSeries Developer Roadmap

How mrc modernized and reinvented its mrc-Productivity Series as the fully portable, Java-based m-Power™ and how you can do it for your business.

This is the story of how mrc personalized the iSeries Developer Roadmap to their needs to modernize their flagship product, the mrc-Productivity Series. And, they were able to do so in a fraction of the time it would have taken following IBM's roadmap to the letter.

All timeframes mentioned in this white paper are in developer-time. One month, for example, means one developer working for one month 100% of the time. So, if you are one developer who can only devote 25% of your time, you'll need to extrapolate from there.

The Basics

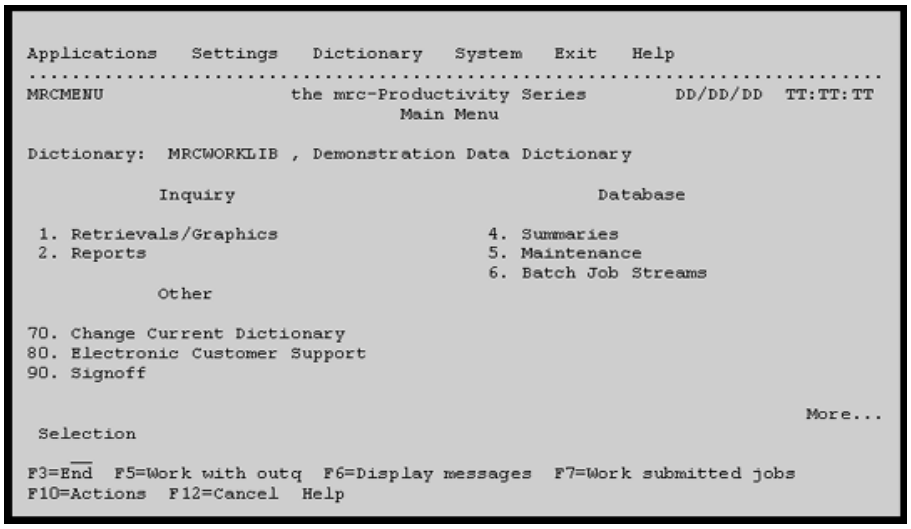
In May of 2002, the top brass at mrc met to discuss the company's overall technical strategy. The meeting was set to determine where its software products were headed and how mrc would begin shaping future development tactics to fit that plan.

Its flagship product, the mrc-Productivity Series, an application development tool that allows developers to rapidly develop custom business applications, still used the traditional text-based menu (**Figure A**) of the iSeries (AS/400). The tool did have Web capabilities allowing its users to create both CGI/RPG and Java-servlet based Web applications from this "old-school" interface. However, it was still fully RPG-based, and the mrc team agreed that it needed to "get with the times."

The Challenge

"We knew we wanted our tool itself to be platform independent and database independent, and we knew Java servlets were the best architecture for platform independent applications. We also knew it would take us a very long time to get to that point if we were going to do it manually," explains Brian Crowley, mrc Director of Development, "So, we sat down

Figure A: *This is what the original mrc-Productivity Series' text-based interface looked like.*



and decided to lay out a plan based on where we were, with interim stages and goals, similar to IBM's iSeries Developer Roadmap. Basically, we took that concept and created a realistic and workable plan to fit our own needs."

The first step was an overall assessment of where they were starting from. This meant an analysis of all the applications running within the tool: which ones were easier, which were more complex, and how they would deal with each.

Because traditional applications or legacy programs often contain multiple screens, it's important to measure how many screens you'll need rather than how many programs you currently have. In the Web world, the rule of thumb generally is one program per screen.

"We knew we had about 263 screens to deal with," explained Crowley, "We measured it by screens because that was going to be the end-result in the Web world: one application per screen." Once mrc counted their screens, they separated them into two groups…easy screens and difficult screens.

"Some programs are just simpler than others," explained Crowley, "For example, lists of items in a database file, or an application for maintaining one file at a time. These are easy screens. Then, there are those that don't fit a standard look, or have more complicated code behind them. These are the difficult screens. When you actually go through your application base, though, you'll probably find that the vast majority of programs fall into the easy screen category."

In mrc's case, this was 209 of the 263 screens, or 80% of the project. This percentage, it should be noted, may be specific to the nature of redeveloping a complex software tool. For most businesses, this will probably be a much higher "easy" percentage…more of a 95% easy to 5% difficult split.

Once mrc's developers had a good idea what they were dealing with, they laid out a plan in two phases.

The Solution: mrc Phase 1

mrc decided to create something their customers needed and wanted: a Java-based Web interface for the mrc-Productivity Series that they could use to develop their Java applications.

IBM's Stage 1: Improve your Productivity

mrc knew the easiest way to accomplish their roadmap end-goal was to use their own tool to do the majority of the modernization. mrc's Java template technology allowed them to quickly write their new screens in server-side Java.

When mrc created the mrc-Productivity Series' Java templates to Enhance the End User Experience, they also taught the tool to

develop applications using a Modular Architecture. This "Create a Modular Architecture" stage separates the user interface from the application logic tier and the database tier or module. This choice to create the templates in this manner allowed them to combine IBM's stages 2 and 3 as they began modernizing.

mrc's templates were built to not only deploy a Graphical User Interface GUI, and separate that interface from the application logic, and database commands into Better Architecture, but it was taught to write Java servlet applications for better portability.

With its better tool, mrc began to put together a combined assault on the next three IBM stages:

IBM's Stages 2, 3, 4: Enhance the End User Experience; Create a Modular Architecture; Integrate Applications

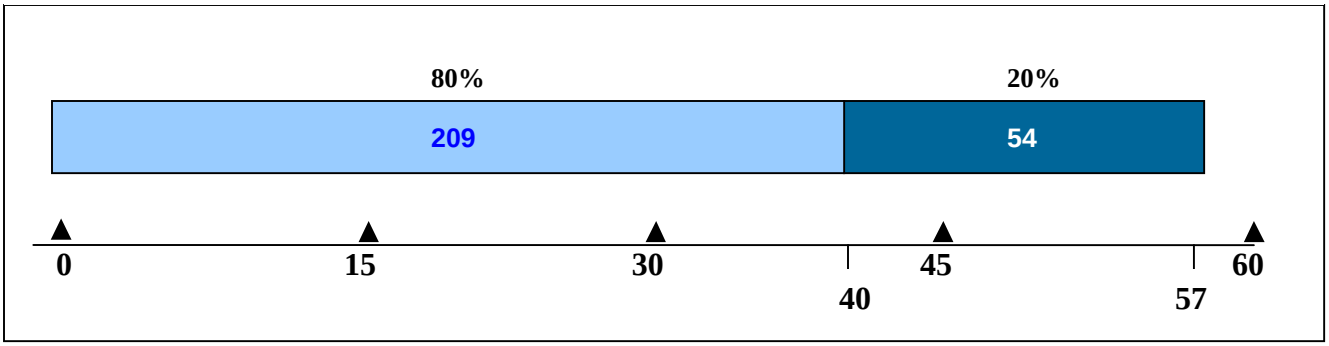
This is where mrc began to create the mrc BED interface (BED stands for Browser-Enhanced Development). This stage starts out right in line with IBM's iSeries Developer Roadmap but mrc is able to incorporate IBM's Better Architecture and Better Portability stages at the same time.

After determining that 209 of its 263 screens requiring modernization were "easy screens" such as lists, reports, simple maintenance apps, mrc began redeveloping its screens in Java servlets by using its "Better Tool" mrc-Productivity Series' with its old text-based menu.

This development process took one mrc developer just *four months* to do the first 80% of the mrc-Productivity Series' BED interface.

Admittedly, the remaining 54 screens making up the other 20% of Phase 1 were a little more difficult. (mrc considered difficult screens to be those with multiple lists, complex calculations and complicated SQL logic, maintaining un-related data at once, etc.) This second part of Phase 1 modernization took mrc's developer 9 months to complete.

Figure B: *This diagram represents mrc's Phase 1, if mrc's developer had elected to develop* ***without the mrc-Productivity Series..***



If mrc had followed IBM's roadmap manually to get to this point, it would have taken about 40 months to get through the first 80% and another 17 months to get through the second half, and this is without any portability. IBM's total project time: 57 months or over 4 and a half years. (**Figure B**)

By using the mrc-Productivity Series, they were able to combine stages, and develop 80% of their Java applications in four months with a Better User Interface and Better Architecture, following up with the remaining 20% of their applications (54 screens) in the next 9 months. mrc's total project time: 13 months or 1 year and 1 month. (**Figure C**)

That's a time savings of 90%.

At this stage, the majority of programs were fully portable, and could access any database and run in any environment. However, there were still some RPG programs that were being called behind the scenes.

For most businesses, this would be the end of the road... If there is no immediate desire to change your platform or database, getting to this stage will usually do the trick. Users are happy to have a fast and efficient Web interface, which can remove most of the immediate pressure, allowing you to begin learning Java, or figure out your next step if you choose to move toward full portability.

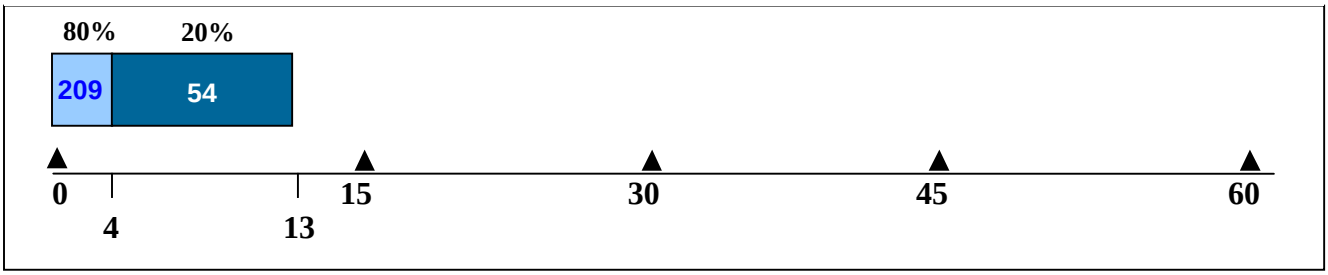
That means getting to your final destination in **1/10 of the time** it would take following each stage of the roadmap. Now imagine that time savings if you are on the 95%-5% split instead!

However, in mrc's case, because these programs were part of a larger product, mrc was still unable to offer a fully portable solution. Let's track mrc's progress and follow their version of the roadmap to its final destination.

The Solution: mrc Phase 2

Phase 2 is all about tackling the remaining platform and database-independence, or full portability, of the mrc-Productivity Series itself.

Figure C: *This diagram represents mrc's Phase 1,* ***the actual timeline****. It took one mrc developer just four months to modernize 80% of mrc's applications using the mrc-Productivity Series…and the total development time to get to mrc's BED interface took just 13 months.*



IBM Stage 4: Integrate Applications …continued

For this last little bit, to get better portability, you will need to know Java, or hire someone who does. The difference is, by using mrc's tool, when you get to this stage you have just a fraction of manual coding to do now, making a world of difference to both timeframe, and bottom line.

By the time mrc got to this point, the majority of their application logic was written in Java with the tool, but there were still some significant RPG-based logic that our newly modernized screens were calling behind the scenes as stored procedures. That meant the application was still tied to the iSeries, and was not fully portable.

For example, in mrc's case, the code-generator that is the workhorse of the mrc-Productivity Series was still written in RPG entering this stage.

The Java-based BED interface was still able to use this RPG-based code generator throughout the modernization process without interruption because the generator program is called as an external object.

However, in order to make their software truly platform and database-independent, this RPG component would need to be manually written in Java. Once written, this new Java-based code generator program seamlessly replaces its RPG counterpart behind the scenes…with no one the wiser.

mrc counted 35 RPG-based stored procedures that needed to be rewritten in Java logic, and some back-end batch logic that needed to be re-created as well. But, basically, this stage was handled as a development task list.

This stage took mrc 19 months. A word of warning…this was with an experienced Java developer, so if you are new to Java, or you are hiring a consultant, you will need to factor the experience level of the Java developer into the timeframe.

mrc's final result? m-Power™. Without RPG calls, m-Power™ is fully portable, and can run anywhere Java runs and access any database. That means it could run on the iSeries, the xSeries, the zSeries, the LMNOPSeries…your PC, you name it.

IBM Stage 5: Integrate Business Processes.

Because mrc develops applications in J2EE automatically, it has given itself, and its customers, a real advantage when it comes to integrating business processes. This distinction paves the way for using Enterprise JavaBeans or EJBs, reaching IBM's last roadmap stage.

The Value:

mrc, in modernizing its mrc-Productivity Series, and developing m-Power™ its fully portable development tool, enjoyed three immediate benefits.

First, and foremost, it became easier to use. Developers and users can develop applications remotely, and securely access resulting applications from any Web browser in the world. m-Power and the new mrc-Productivity Series' GUI also made it easier to train users to develop their own applications. Additionally, the benefits of the GUI interface allow the incorporation of live Web demonstrations, links to online manuals, and email to live Web support.

Secondly, it gives your business more control. Its portability means it is no longer reliant on any one hardware vendor, operating system, or database and offers a way for its customer to determine the most cost-effective methods of running the technology side of their business.

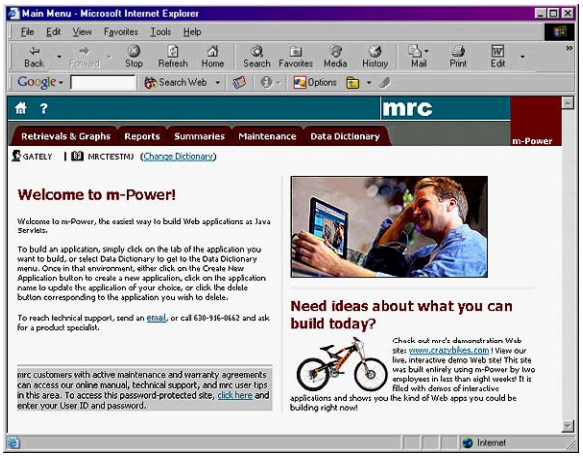
Thirdly, it's faster. Without the additional RPG calls within the program, the Java servlets access database information at lightning speeds, and response-time is unprecedented.

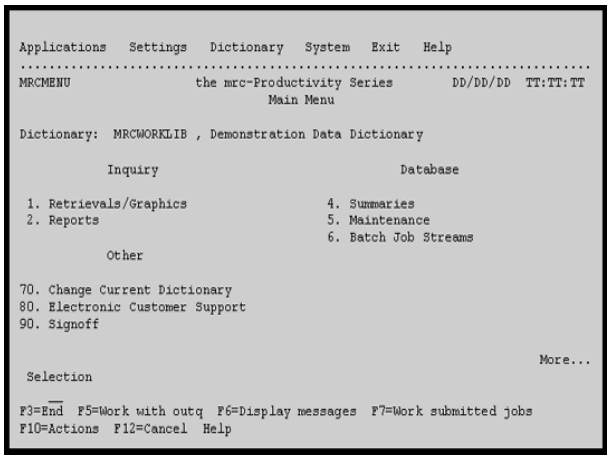
Here are some additional benefits, specific to using the mrc-Productivity Series in the iSeries Developer roadmap:

Save time and money: By using the mrc-Productivity Series to take care of modernizing the vast majority of your applications…even if you need to take the time to learn Java, or hire a Java developer…you're only paying for a fraction of the applications you would have had to manually re-develop otherwise. And, by speeding you through the roadmap, you are saving years in development time through mrc's template-based code generator.

Relieve pressure: Because everything is initially left intact behind the pretty Java user interface, users and management tend to relax their pressure. Then, rewriting the remaining 20% can just be addressed as a simple IT

Figure D: *This figure is the new graphical Java-based Web interface of m-Power™'s menu, compared here, next to the original text-based menu found in Figure A.*





checklist. Although these changes are still crucial in the long term, it allows IT departments to divide and conquer, and tends to reduce the intense pressure modernization plans can instill.

Reduce risk: You can safely modernize the majority of your applications without breaking anything, and without knowing Java or performing any manual coding. And, you can quickly deliver to your users the Web interface that they have been demanding. It is an entirely new system, built in Java, using n-tier architecture. However, the underlying code used, or the program logic, is time-tested and bug-free.

For example, mrc had RPG code like edit checks and custom calls that they had to address. In order to get to market faster, they didn't want to have to take the time to re-write custom code, and then take the additional time testing it when it worked perfectly well as it was.

Additionally, because the resulting Java application's source code is fully modifiable, developers are able to make any custom changes they need without being tied to the tool.

Built-in Java education: mrc has a hidden bonus in that you gain real exposure to Java, its structure, and how it works. The accessibility you have to the underlying Java source code can start you and your development team well on your way to learning Java as you use it, giving you yet another added advantage on the road to modernization.

In the end, there is no silver bullet. Whatever path you choose to take through the roadmap will necessarily be unique, and custom to your business needs. By developing its own plan within the parameters of the roadmap, mrc was able to modernize its complex development software product in record time. This story should simply serve as a case study of how this iSeries-based company managed to successfully take the concept of the iSeries Roadmap through to fruition, and provide some insight as to how you can do the same.

If you'd like help getting started, mrc offers free consultations, and can help you determine how many screens you'll need and what categories these screens will fall into to help you determine your business's percentage and starting point on the roadmap.

Just visit here to get started:
http://www.mrc-productivity.com/forms/M21070GC.mrc

Endnotes:

[i] "iSeries Developer Roadmap" http://www-1.ibm.com/servers/eserver/education/cust/iseries/paths/j2ee/ests_isli_j2ee_index.html

[ii] www.iseriesnetwork.com/*/iSNblogs/wdsc_survivor*

[iii] Hoffman, Sharon. "An Independent Analysis of the iSeries Developer's Roadmap." 2004. ftp://ftp.software.ibm.com/common/ssi/rep_wh/n/ISW00197USEN/ISW00197USEN.PDF

[iv] "iSeries Developer Roadmap" http://www-1.ibm.com/servers/eserver/education/cust/iseries/paths/j2ee/ests_isli_j2ee_index.html

[v] "iSeries Developer Roadmap" http://www-1.ibm.com/servers/eserver/education/cust/iseries/paths/j2ee/ests_isli_j2ee_index.html

[vi] Hoffman, Sharon. "An Independent Analysis of the iSeries Developer's Roadmap." 2004. ftp://ftp.software.ibm.com/common/ssi/rep_wh/n/ISW00197USEN/ISW00197USEN.PDF

[vii]"iSeries Developer Roadmap" http://www-1.ibm.com/servers/eserver/education/cust/iseries/paths/j2ee/ests_isli_j2ee_index.html

[viii] Hoffman, Sharon. "An Independent Analysis of the iSeries Developer's Roadmap." 2004. ftp://ftp.software.ibm.com/common/ssi/rep_wh/n/ISW00197USEN/ISW00197USEN.PDF

[ix] http://www.potentialsmag.com/potentials/search/article_display.jsp?schema=&vnu_content_id=1000740214